UNITED STATES PATENT APPLICATION

FOR

# METHOD AND APPARATUS FOR PARALLEL TRUNKING OF INTERFACES TO INCREASE TRANSFER BANDWIDTH

Inventors:

Ariel Hendel
Leo Hejza
and
Howard Frazier

prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 Wilshire Blvd., 7th Floor
Los Angeles, California 90025-1026
(408) 720-8598

Attorney Docket No.: 082225.P2170

10

13846

101A

08/813647





# METHOD AND APPARATUS FOR PARALLEL TRUNKING OF INTERFACES TO INCREASE TRANSFER BANDWIDTH

## FIELD OF THE INVENTION

The present invention relates to the field of computer networks. More specifically, the present invention relates to a method and apparatus for parallel trunking of interfaces to increase transfer bandwidth between network devices.

## BACKGROUND OF THE INVENTION

Local Area Networks (LANs) following the IEEE 802 Standard Architecture are network environments that interconnect end-nodes using various types of network elements. Each of these network elements is subject to its own set of configuration and topology guidelines. These sets of guidelines and rules are intended to deliver a uniform and well defined data link layer behavior over which upper network layers can operate. Examples of this behavior include the emulation of loop-free broadcast domains where any node may communicate with any other node with no prior signaling required, where packet ordering is preserved between any pair of end-nodes, and where every packet is delivered to its destination no more than once.

Figure 1 illustrates a plurality of end-nodes interconnected by a network. End-nodes 110-113 are typically computers capable of sourcing packets to be delivered over the network 120 to other end-nodes. The end-nodes 110-113 are also capable of sinking packets sent to them by other end-nodes in the network 120. Routers are also considered end-nodes for the purposes of this description. Typical network elements used to build LANs are repeaters (hubs) and bridges. In some cases bridges are designated as LAN switches. The IEEE 802.1d Standard for transparent bridges (LAN switches) and IEEE 802.3 Standard for repeaters and end-nodes provide the topological rules that allow the data link layer behavior described.

Any one of the end-nodes 110-113 may send packets to any other end-node connected to the network 120. Packets are delivered at most once per destination. A sequence of packets sourced at one of the end-nodes 110-113 is seen in the same order at a destination end-node connected to the network 120.

Figure 2 illustrates a network implementation. The network 200 includes a plurality of repeaters 210 and 211 and switches 220-222. As a rule, links or interfaces connected to repeaters have the same nominal speed or bit rate and links or interfaces connected to switches can have dissimilar speeds or bit rates. An additional property of the switches 220-222 is traffic isolation. Traffic isolation consists of forwarding packets of data only through the links or interfaces where the destination may reside. This property was deliberately defined to increase the overall network capacity. In order to accomplish traffic isolation, switches 220-222 must know which end-nodes 230-236 are reachable via each link or interface. Bridges or switches complying with IEEE 802.1d automatically learn this information with no assistance from the end-nodes 230-236 or any other external agent, and are called transparent bridges.

Figure 3 illustrates a transparent bridge. Bridge 310 is connected to end-nodes 311-313. A packet from end-node 311 to end-node 312 is not sent to the link where end-node 313 resides because of the property of traffic isolation. The bridge 310 learns the location of the end-nodes 311-313.

Network capacity is central to the function of a network. Traditionally, there have been two approaches to increasing network capacity. The first approach involves partitioning the network. The partitioning approach uses switches, bridges, and routers to replace a network of repeaters. By replacing the network of repeaters with more sophisticated hub devices, the flow of packets in the network is better managed and system performance is increased. The second approach involves providing faster link

alternatives. By implementing links that can support higher bandwidth, the network capacity is increased.

Nevertheless, at any given point in time, the choice of link speeds (10, 100, 1000 Mbps) may not match up very well with the amount of sustained throughput that a particular device can support. When switches and high performance routers are used to interconnect multiple links of a given speed, there is a clear need for the inter-switch or inter-router link to be able to support at least some aggregation of the links. If new hardware is required to utilize a newer, higher speed, increased bandwidth network, the utilization of the newer network may not be as attractive from a cost standpoint. Furthermore, the cost associated with implementing a newer network with more sophisticated hubs or faster links may also not be attractive from a cost standpoint.

Thus, what is needed is a method and apparatus for increasing data transfer bandwidth between network devices, such as end-nodes and switches.

SUMMARY

A method for interconnecting a first device and a second device in a network is described. The first device and the second device are connected to a plurality of interfaces. The plurality of interfaces emulate a single high-speed interface.

A method for creating a multiple interface connection is disclosed. A first identifier is assigned to a first interface and a second interface at the first device. A path between the first device to the second device is identified with the first identifier.

A second method for creating a multi-interface connection is disclosed. A first device is connected to a plurality of interfaces. The plurality of interfaces emulate a single high-speed interface.

A network is disclosed. The network includes a first device and a second device. A first interface is coupled to the first device and the second device. A second interface is coupled to the first device and the second device. The first interface and the second interface emulate a single high speed interface. According to an embodiment of the present invention, the first interface and the second interface are assigned an identifier that identifies a path between the first device and the second device.

A network device is disclosed. The network device includes a first port that is connected to a first interface. The network also includes a second port that is connected to a second interface. A trunking pseudo driver is coupled to the first port and the second port. The trunking pseudo driver allows the first interface and second interface to emulate a single high-speed device.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not by way of limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

Figure 1 illustrates a plurality of end-nodes interconnected by a network;

Figure 2 illustrates a network implementation;

Figure 3 illustrates a transparent switch;

Figure 4 illustrates a block diagram of a system which may be programmed to implement the present invention;

Figure 5 illustrates a network implementing an embodiment of the present invention;

Figures 6a illustrates a first device and a second device connected to a trunk connection;

Figure 6b illustrates the first device and the second device as a server and a switch;

Figure 6c illustrates the first device and the second device as switches;

Figure 6d illustrates the first device and the second device as servers; and

Figure 7 illustrates a software embodiment of a server interface according to an embodiment of the present invention.

DETAILED DESCRIPTION

A method and apparatus for high speed data transfer is disclosed. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

COMPUTER SYSTEM OVERVIEW

Referring to Figure 4, the computer system upon which an embodiment of the present invention can be implemented is shown as 400. Computer system 400 comprises a bus or other communication device 401 that communicates information, and a processor 402 coupled with bus 401 that processes information. System 400 further comprises a random access memory (RAM) or other dynamic storage device 404 (referred to as main memory), coupled to bus 401 that stores information and instructions to be executed by processor 402. Main memory 404 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 402. Computer system 400 also comprises a read only memory (ROM) and/or other static storage device 406 coupled to bus 401 that stores static information and instructions for processor 402. Data storage device 407 is coupled to bus 401 and stores information and instructions. A data storage device 407 such as a magnetic disk or optical disk and its corresponding disk drive can be coupled to computer system 400. Network interface 403 is coupled to bus 401. Network interface 403 operates to connect computer system 400 to a network (not shown).

Computer system 400 can also be coupled via bus 401 to a display device 421, such as a cathode ray tube (CRT), for displaying information to a computer user. An

input device 422, including alphanumeric and other keys, is typically coupled to bus 401 for communicating information and command selections to processor 402. Another type of user input device is cursor control 423, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 402 and for controlling cursor movement on display 421. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane.

Alternatively, other input devices such as a stylus or pen can be used to interact with the display. A displayed object on a computer screen can be selected by using a stylus or pen to touch the displayed object. The computer detects the selection by implementing a touch sensitive screen. Similarly, a light pen and a light sensitive screen can be used for selecting a displayed object. Such devices may thus detect selection position and the selection as a single operation instead of the "point and click," as in a system incorporating a mouse or trackball. Stylus and pen based input devices as well as touch and light sensitive screens are well known in the art. Such a system may also lack a keyboard such as 422 wherein all interface is provided via the stylus as a writing instrument (like a pen) and the written text is interpreted using optical character recognition (OCR) techniques.

The present invention is related to the use of computer system 400 to facilitate high speed data transfers via a trunk connection. According to one embodiment, facilitating high speed data transfers via a trunk connection is performed by computer system 400 in response to processor 402 executing sequences of instructions contained in memory 404. Such instructions may be read into memory 404 from another computer-readable medium, such as data storage device 407. Execution of the sequences of instructions contained in memory 404 causes processor 402 to facilitate high speed data transfers via the trunk connection, as will be described hereafter. In

alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the present invention. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

## NETWORK OVERVIEW

The present invention increases the capacity of individual network links or interfaces that do not have repeaters at either end while preserving the guidelines specified by IEEE 802 as perceived by other end-nodes and network elements in the network. The capacity is increased by connecting an arbitrary number of similar links or interfaces in parallel. This approach is useful whenever increasing the raw speed of the existing link is not technically or economically feasible, or when the physical proximity makes parallel links more appealing than changing the link to faster interfaces and media types.

Figure 5 illustrates a network according to an embodiment of the present invention. The network 500 includes a plurality of repeaters 510 and 511 and a plurality of switches 520-522. Trunk 540 connects the switch 521 with the switch 522. Trunk 541 connects switch 522 with end-node 533. Trunk 540 and trunk 541 include a plurality of links or interfaces connected in parallel. Connecting a plurality of links or interfaces in parallel increases the capacity in the path between two devices. The present invention implements trunks while preserving the properties of a IEEE 802 network by preserving a behavior that is transparent to and inter-operable with all other end-nodes and network elements that do not participate in the trunk.

Preserving the properties of IEEE 802 requires addressing the following problems:

1) A conventional switch connected to the same media access control (MAC) address over more than one link would only use one of these lines (the one learned last);

2) A trunk is a loop, and loops between switches can be broken by 802.1d;

3) Parallel paths may cause packet re-ordering;

4) A conventional 802.1d switch delivers multiple copies of a packet when an end-node is multi-honed;

5) A conventional 802.1d switch could loop packets back over the other links of a multi-honed end-node.

Each device connected to the trunks 540 and 541 has a trunking layer. The trunking layer is responsible for load balancing to determine which link or interface to use to transmit a given packet of data. Load balancing is applicable to both end-nodes and switches. The layer has additional duties in the cases of switches, including eliminating looped packets, suppressing multiple copies of a packet, and treating the entire trunk connection as a single logical port in its topology database. The trunking layer uses the same MAC address for all the links or interfaces on a trunk to maximize the transparency towards the protocol stack executing in the end-nodes. By using the same MAC address, a single IP address may be associated with the entire trunk.

Figure 6a illustrates a first device 610 and a second device 620 connected to a trunk 630 that includes a plurality of links or interfaces 631-633. The first device 610 and the second device 620 may be the switch 521 and the switch 522 connected to the trunk 540 or the switch 522 and the end-node 533 connected to the trunk 541 in Figure 5. Alternatively, the first device 610 and the second device 620 may be an end-node, such as a server, client, or router, or a switch as illustrated in Figures 6b-6d. Figure 6b illustrates an embodiment of the present invention where the first device 610 is a server and the second device 620 is a switch. Switch 620 is coupled to a plurality of client segments labeled 641-645. The server 610 may be implemented by the computer system 400 illustrated in Figure 4. Figure 6c illustrates an embodiment of the present invention where the first and second devices 610 and 620 are switches. The switch 610

is coupled to a plurality of client segments labeled 651-655 and the switch 620 is coupled to a plurality of client segments labeled 641-645. Figure 6d illustrates an embodiment of the present invention where the first and second devices 610 and 620 are servers.

Figure 7 illustrates a software embodiment of an interface of a computer system connected to a trunk according to an embodiment of the present invention. Upper layers 710 represent any application program that produces or consumes data in the computer system. IP 720 represents an Internet Protocol (IP) layer that makes IP-addressed packets possible. Network device driver 740 represents a layer in the operating system that facilitates communication between hardware and the operating system. Device Units 751-753 represent the physical hardware connected to each of the interfaces 631-633. According to an embodiment of the present invention, interfaces 631-633 are connected to a network device via ports. The trunking pseudo driver 730 resides between the IP layer 720 and the network device driver 740 and contains the trunking layer described above. The trunking pseudo driver 730 splits data in the transmit path and merges data in the receive path of the interfaces 631-633. It should be appreciated that the trunking pseudo driver may be implemented by any known circuitry in a hardware environment.



TRUNK CONNECTION ASSIGNED A IDENTIFIER

Referring back to Figure 6a, the plurality of interfaces 631-633 operate to provide a high bandwidth connection between the first device 610 and the second device 620. The physical interfaces 631-633 share a common source device and destination device with each other. The number of interfaces that are implemented may be any number greater than two and dependent on the bandwidth requirement of the network 200; and "trunk" as used herein refers to any such multiple-interface connection, i.e. a

connection having at least two links or interfaces. The plurality of interface 631-633 are assigned an associated identifier that identifies the connection between the first device 610 and the second device 620. For end-nodes, the identifier may be a logical name such as a media access control (MAC) address or an Internet Protocol (IP) address. For switches, the identifier may be a grouping identifier with local significance only. End-nodes connected to the trunk 630 will associate all the interfaces 631-633 of the trunk 630 by its identifier. According to an embodiment of the present invention, interfaces 631-633 are Ethernet interfaces, but may be any suitable network interface such as an Intranet interface (such as LAN) or Internet interface. Interfaces 631-633 may be homogeneous, having identical physical layer and media access control layer characteristics, or non-homogeneous. According to a preferred embodiment of the present invention, the physical and media access control layers for each interface are full duplex.

## LOAD BALANCING

In order to maximize the throughput rate of data transmitted on the trunk 630, the first device 610 and the second device 620 select one of the interfaces 631-633 in the trunk 630 and uses the selected interface to transmit data. Load balancing in end-nodes typically involves utilizing state information regarding previously sent data, and the status of output queues corresponding to the plurality of interfaces 631-633 in selecting an interface to transmit present data. State information regarding previously sent data is available to the end-node because the software generating the data is running in the same environment as the trunked end-node interface. The depth of the output queue is used as a metric for determining how busy a physical interface is.

The temporal ordering of the packets must be preserved when a stream of packets is to be transmitted from one end-node to one or more end-nodes. In order to

satisfy the temporal ordering, the end-node will attempt to ensure that all of the packets associated with a particular transport layer datagram are enqueued on the same network device transmit queue.

According to one embodiment of the present invention, the end-node inspects the header of each packet and uses the information to associate each packet with a particular connection. The end-node keeps a small cache of MAC or IP destination addresses associated with each network interface. When the IP hands the end-node a packet, the end-node checks the cache to see if it has recently transmitted a packet to this destination address. If it has, the end-node will enqueue the packet on the same interface that it enqueued the last packet to this destination address. If the destination address has not been transmitted to recently, the pseudo driver can either enqueue the packet on the last busy transmit queue or the emptiest queue, or the next available queue in a round robin fashion. The end-node updates the cache for that queue with the new destination address. In the situation where the server is sending data to one client, this technique would ensure that all packets to the client travel over the same interface on the trunk.

Load balancing in switches typically involves selecting an interface based on the source address of the packet, or of the packet's port of arrival. The interface selected could, for example, be looked up on a table or calculated using a deterministic algorithm. This scheme results in a static load balancing function that forwards most of the traffic along the same physical interface. As an improvement, it is possible to have a dynamic mapping function and still maintain frame ordering, given that the function changes are slower than the output queue transit times. For instance, the mapping for a given source address can be determined at the time the first packet with the source address is seen, and eventually aged when the source address is not seen for a period of time. By considering both the source address and the port of arrival, the dynamic

mapping function reduces the number of pathological cases. For example, if the traffic is spatially dominated by a particular input port, considering the source address helps spread its traffic, and conversely the port of arrival helps distribute traffic dominated by a small number of addresses in particular if more than one trunk connection exists in the switch.

In an alternate embodiment of the present invention, the mapping function separates traffic according to priority or whether the traffic is bandwidth managed. A priority based mapping function is desirable when packet order preservation is not necessary.

Referring back to Figure 6a, it should be appreciated that the load balancing techniques implemented on device 610 and device 620 do not have to be the same in order to implement the trunk 630. It should also be appreciated that the interfaces 631-633 may be used to transmit data packets bi-directionally.

## LOOP PREVENTION

Prevention of frame duplication is achieved at the switch ~~is achieved~~ by treating the set of trunked ports on the switch as if they were a single port with a separate queue per physical interface. Forwarding is performed to only one of its queues. Thus, packets of data with broadcast, group, multicast, or unknown unicast address are not replicated or duplicated across the interfaces 631-633 of the trunk 630. Data transmitted through an interface of the trunk 630 are also not sent back through another interface of the trunk 630.

In the foregoing description, the invention is described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modification and changes may be made thereto without departing from the broader spirit and scope of

the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

Additional details regarding implementations of the present invention are attached in the Appendix.

# Simple Trunking Model (STruM)

**Howard Frazier, Leo Hejza, Ariel Hendel**
**SMCC/NPG**
**March 3, 1997**

## 1. Introduction

Ethernet has learned a new trick in the past couple of years. It has learned how to scale it's link speed by a factor of ten. 10BASE-T begot 100BASE-T, which in turn will be scaled up to one gigabit per second with 1000BASE-T. The Ethernet community has deliberately and emphatically rejected the idea of specifying link speeds in anything other than multiples of 10. Proposals have been made, and shot down in flames, for intermediate link speeds between 10 and 100 Mbps, and they have recently been made again in the context of Gigabit Ethernet, and they have been met with the same reception. No one who sells networks or computers wants to confuse the marketplace by producing multiple, non-interoperable, intermediate link speeds, and everyone seems to feel more comfortable when the link speed can be expressed as power of the number of our fingers or toes.

Never the less, at any given point in time, the choice of link speeds (10, 100, 1000 Mbps) may not match up very well with the amount of sustained throughput that a particular device can support. Virtually all multi-processor servers shipped today can sustain greater than 100 Mbps aggregate network transfer rates. Furthermore, when switches and high performance routers are used to interconnect multiple links of a given speed, there is a clear need for the inter-switch or inter-router link to be able to support at least some aggregation of the links. The next power of ten increase in link speed may not be an attractive choice from a cost standpoint unless the utilization of the higher speed link is going to be greater than 40 to 50 percent. Kicking up the link speed also requires new hardware.

For the purposes of this discussion, trunking will be defined as the ability to combine multiple parallel physical links into one logical channel. We will limit ourselves to trunks in which the physical links share a common source, and a common destination. We will further limit ourselves to trunks in which each of the links (or "segments" of the trunk) have identical physical layer and media access control layer characteristics. We have paid particular attention to the way IP packets will be transported over these trunks, but we don't believe that the model described herein is in any way limited IP networks.

The remainder of this paper will describe a set of rules that the equipment on each end of the trunk must agree to. The rules can be applied to either end stations (DTEs, such as computers of any classification) or network infrastructure components (specifically switches). The rules are symmetric, which is to say that both ends are subject to the same rules.

## 2. Rules

1. A trunk may have any number of segments, but all segments must have identical physical layer and media access control layer characteristics
2. Each segment of the trunk shares a common source and a common destination with the other segments of the trunk
3. Temporal ordering of the packets transported across a given segment of the trunk must be preserved throughout the network, subject only to loss due to bit errors
4. Temporal ordering of the packets transported across different segments of the trunk must not be assumed
5. Packets must not be replicated or duplicated across the segments of a trunk. This includes broadcast and multicast packets
6. Broadcast and multicast packets transmitted through a segment of the trunk must not be "echoed" or "looped-back" to the sender over the other segments of the trunk
7. The model assumes full duplex operation at the physical and media access control layers for each segment. Half duplex operation, using CSMA/CD, is neither supported nor desired
8. End stations connected to trunks will associate a single 48 bit IEEE MAC address with all segments of the trunk.
9. Load balancing across the segments is not assumed to be perfect. Each end of the trunk will attempt to load balance across the segments to the best of its ability, subject to all of the forgoing rules

These rules do not addresses the configuration, setup, management, or maintenance of trunks, nor do they address failure detection or recovery. It is assumed that the physical layer will provide some indication if a particular segment of the trunk fails, and that each end of the trunk will monitor whatever status is provided by the physical layer, and take whatever action is deemed appropriate Configuration and setup are assumed to be performed via manual operations specific to each implementation.

## 3. Proposals for load balancing

The diagrams in Figure 1 through Figure 3 may assist the reader in understanding the proposals

Figure 1 shows a trunk connection between a server (as a specific example of a DTE) and a switch. The example shows a trunk with 3 segments, labeled A, B, and C. The switch is also connected to several client segments, labeled a, b, etc. In this configuration, it is

possible to replace the server with other types of equipment, such as a router, or a high performance workstation, or a printer, to list a few examples.

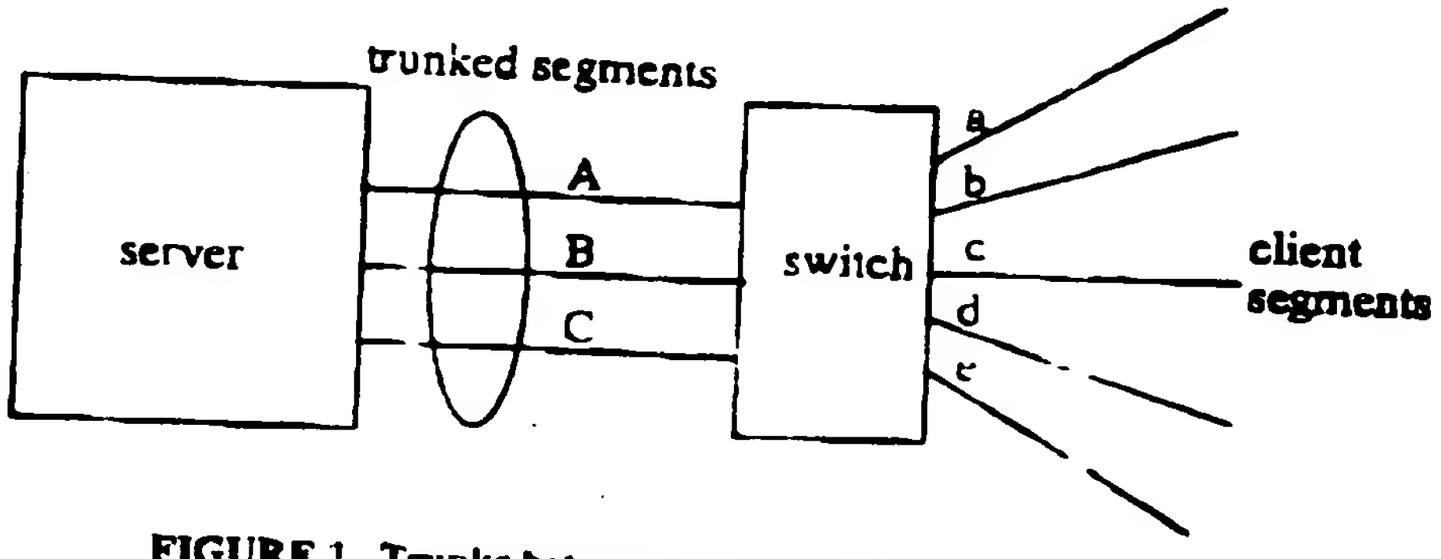


**FIGURE 1. Trunks between servers and switches**

Figure 2 shows a trunk used as a connection between two switches. As in the previous example, there is no special significance to the number of segments which make up the trunk in Figure 2. The trunk could just as easily be made of two or four or practically any number of segments, depending on the amount of bandwidth required.

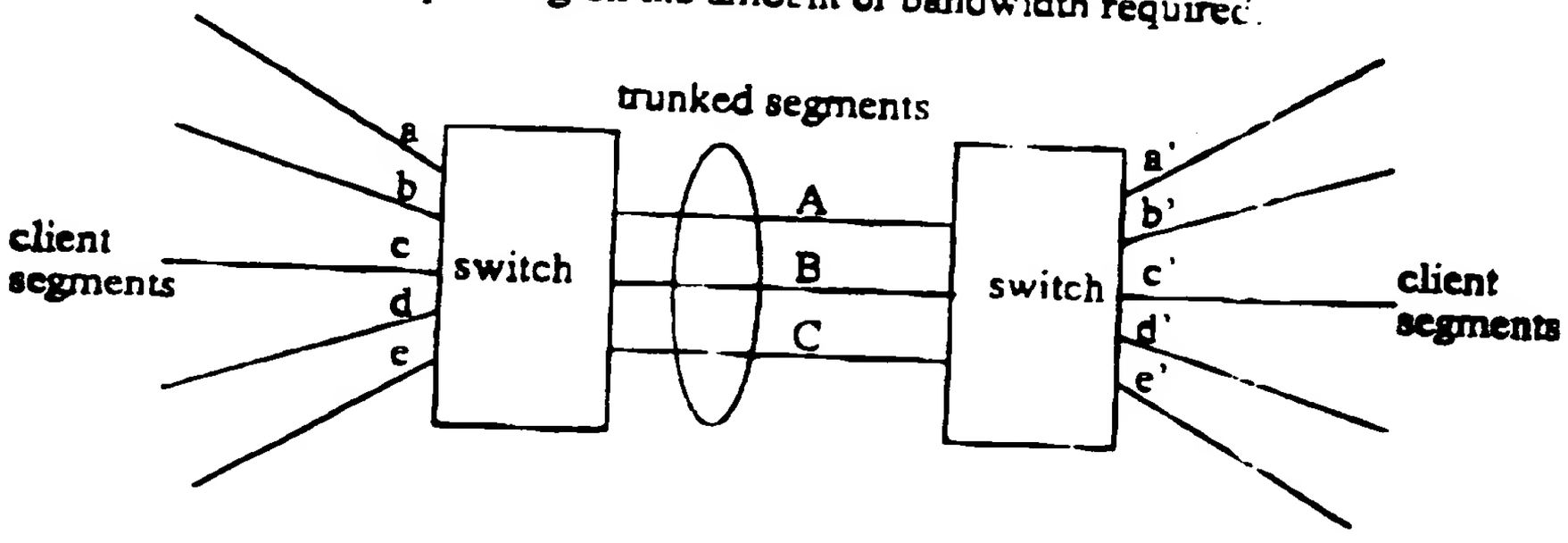


**FIGURE 2. Trunks between switches**

Figure 3 shows a trunk employed between two servers. Once again, either or both of the servers could be replaced with some other type of equipment, such as a router.

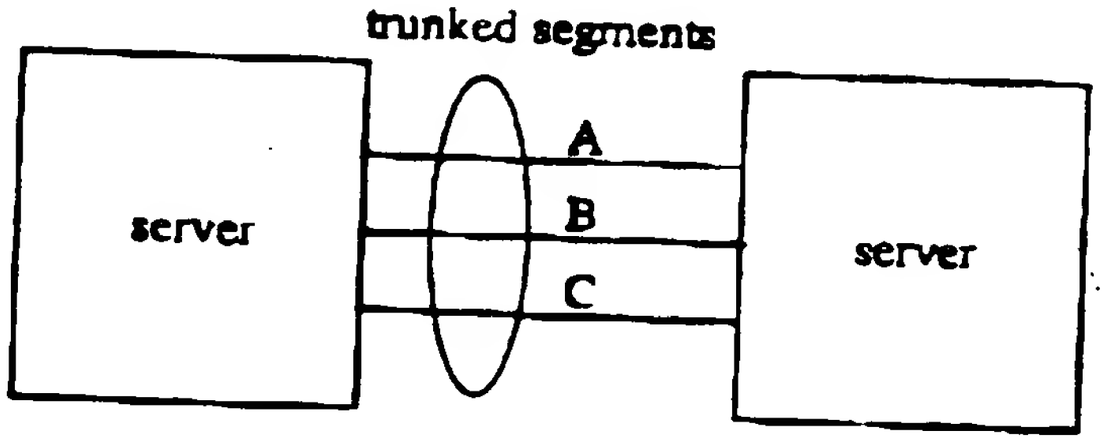


**FIGURE 3. Trunks between servers**

## 3.1 Load balancing in servers

An important goal of the model is to make the trunk appear like a single high bandwidth interface from the viewpoint of the server's protocol stack. In addition, all clients which communicate with the server via the trunk should have a consistent view of the server's identity (MAC and IP host addresses). Load balancing by managing the Address Resolution Protocol (ARP) tables in the clients has been considered and rejected because it would dramatically increase the number of ARP frames emitted by the server, and would require a significant amount of added functionality in the server's ARP implementation.

In order to satisfy Rule # 4, "Temporal ordering of the packets transported across different segments of the trunk must not be assumed"the server must ensure that all packets of any sequence of packets which requires temporal ordering are transmitted over the same segment of the trunk.

It should be noted that transport protocols generally can recover from situations where packets arrive out of order, but that this generally entails a significant degradation in throughput, because out of order reception is handled as an exception, and is not optimized. Therefore, the server load balancing mechanism should be designed to take advantage of Rule # 3, "Temporal ordering of the packets transported across a given segment of the trunk must be preserved throughout the network, subject only to loss due to bit errors".

At this point, it might be helpful to introduce a diagram which shows the software components of a trunked server interface.

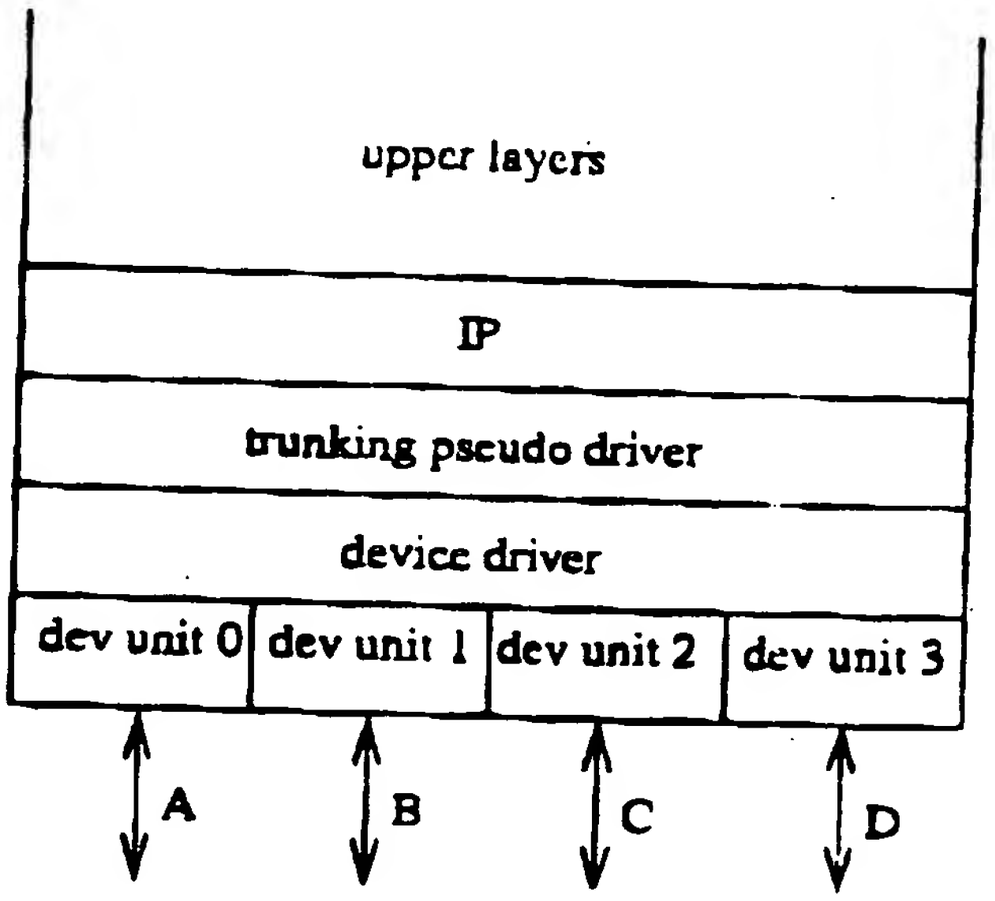


**FIGURE 4. Software components of a trunked server interface**

A "trunking pseudo driver" is introduced between the IP protocol layer and the network device driver. The function of the pseudo driver is to act as a demultiplexor in the transmit path, and a multiplexor in the receive path. In order to satisfy the rules regarding temporal ordering, the pseudo driver will attempt to ensure that all of the packets associated with a

particular transport layer datagram are enqueued on the same network device transmit queue. This assumption is made on the basis that ordering within a datagram is sufficient, and that ordering between datagrams is unnecessary.

It would be quite difficult for the pseudo driver code to inspect the headers of each packet and attempt to associate them with a particular datagram, though one can imagine that this could be accomplished given an arbitrarily fast processor to execute the code. As a first approximation, the authors feel that it would be sufficient to keep a small cache of MAC (or IP) destination addresses associated with each network interface (each segment of the trunk). Thus, when IP hands the pseudo driver a packet, the pseudo driver checks the cache to see if it has recently transmitted a packet to this DA. If it has, the pseudo driver will enqueue the packet on the same interface that it enqueued the last packet to this DA. If this DA has not been transmitted to recently, the pseudo driver can either enqueue the packet on the least busy transmit queue (the emptiest queue), or the next available queue in a round robin fashion. What ever queue it selects, the driver must update the cache for that queue with the new DA.

In the degenerate case in which a server is talking to one and only one client, this technique would ensure that all packets to that client travel over the same interface, and hence the same segment of the trunk. Why do we call this load balancing? Because it works much better in a non-degenerate case, and will do a good job of ensuring ordered delivery if we can safely assume that the degree of interleaving of packets to different DAs between IP and the network driver is of the same order as the number of processors in a given server. As a first guess, the depth of the cache should be equal to roughly twice the number of processors in a given server. The deeper the cache, the more casual the updating to the cache can be. Experimentation to derive the optimal value for the depth of the cache, and to explore the trade-offs between caching layer 2 and layer 3 addresses is warranted.

## 3.2 Load balancing in switches

Several switch load balancing mechanisms are possible for forwarding packets into a trunk. The set of load balancing guidelines listed below apply to both switch to switch and switch to server trunks. They ensure that the switch behavior is consistent with conventional bridging guidelines.

1. No frame misordering for a given priority level between a given MAC source and destination.
2. No frame duplication.
3. Transparent to protocols operating above the MAC layer

A natural approach to load balancing is to emulate a faster link by keeping all trunk segments equally busy, possibly by using the corresponding output queue as the metric for how busy the segment is. As long as the links implement flow control, the output queue length is a good end to end proxy for the segment utilization (without flow control, a high segment load is not necessarily reflected in the state of the output queue due to packet loss on the receive queue at the other end).

Deciding for every packet which segment to use, based solely on queue length, might lead to the frame misordering prohibited by the first guideline. If the decision is only a function of the source address of the packet, or of the packet's port of arrival then the first guideline is always satisfied. This scheme however results in a static load balancing function, and the trunk effectiveness depends on the distribution of the traffic sources. While we anticipate that large number of traffic sources would result in acceptably even distributions, it is still possible to end up with configurations where the mapping function forwards most of the traffic to the same segment.

To handle these cases, It is possible to have a dynamic mapping function and still maintain frame ordering, as long as the function changes are slower than the output queue transit times. For instance, the mapping for a given source address can be determined at the time the first packet with the source address is seen, and eventually aged when the source address is not seen for a period of time.

Having the mapping function consider both the source address and the port of arrival reduces the number pathological cases. For example if the traffic is spatially dominated by a particular input port, considering the source address helps spread its traffic, and conversely the port of arrival helps distribute traffic dominated by a small number of addresses (servers or routers) in particular if more than one trunk exists in the switch.

Prevention of frame duplication is achieved by treating the set of trunked ports as if they were a single port, with a separate queue per segment, and making sure that all forwarding is done to only one of its queues.

Furthermore, for the purposes of other 802.1d functions like learning MAC addresses, filtering frames, and executing the Spanning Tree Protocol (if applicable) trunked ports are also treated as if they were a single port.

So far the discussion was centered around using MAC layer information for load balancing. It is possible for the switch to observe higher level protocol information in order to make better load balancing decisions, as long as the third rule of protocol transparency is followed. Transparency implies that the protocols are not aware nor explicitly cooperate with the switch load balancing function. In addition, for protocols that are not supported or understood by the switch, connectivity must be still guaranteed.

Load balancing based on higher level information is practical for switches that examine Layer 3 headers on a packet by packet basis. Many switches examine Layer 3 headers once, for VLAN configuration for example, and use the corresponding Layer 2 information for packet processing. The potential load balancing merits of this approach were not considered.

### 3.2.1 Other Approaches

The aim of the mapping function could also be different than equally balancing the segments. For example it could separate traffic according to priority, or whether the traffic is bandwidth managed or best effort. A priority based approach is supported by the first

guideline, because packet order preservation is not necessary across different priorities. A priority based approach is straightforward whenever the priority information is well defined at the MAC level (for example if VLAN tags are used).

Restating the main observations, we have shown that the switch behavior is conceptually simple, guided by a set of simple rules along with the particular switch architecture, and can be defined independently of the server load balancing behavior.